

P R E P R I N T

Fracta Language Specification for Fractals

Vincenzo Manto

June 2, 2026

DOI: <https://doi.org/10.5281/zenodo.21755350><https://github.com/VincenzoManto/Fracta>

Abstract

This document defines the formal grammar, lexical structure, and runtime execution semantics of Fracta v1.0, a minimalistic, line-oriented domain-specific language (DSL) designed for the generation of recursive geometric topologies. Fracta maps string-rewriting Lindenmayer Systems (L-Systems) directly into deterministic vector trajectories via a stack-based turtle graphics state machine. The language optimizes syntactic overhead to achieve low cognitive friction and high parse efficiency in resource-constrained or distributed environments.

Keywords: math, Fractals, DSL

Status of This Memo

This specification documents the stable core syntax of Fracta version 1.0. Conforming interpreters and compilers **MUST** implement the complete lexical validation rules and execution state machine outlined herein to ensure platform-agnostic output consistency.

Table of Contents

1. Introduction and Architectural Intent
2. Lexical Structure & Tokenization
3. Core Directives & Structural Syntax
4. Turtle Graphics Alphabet & Execution Semantics
5. Formal Grammar (EBNF)
6. Reference Syntax Blueprint
7. Runtime State Machine & Memory Layout
8. Results

1. Introduction and Architectural Intent

Lindenmayer Systems provide a powerful mathematical framework for modeling plant growth, cellular structures, and space-filling recursive curves. However, implementing these systems programmatically often requires custom data structures or verbose configuration scripts that scale poorly under line-oriented serialization frameworks.

The Fracta language standard bridges this gap by providing an ultra-lean, plaintext tokenization layout. Fracta isolates the genetic elements of a fractal (the axiom and the production rules) and exposes them through a deterministic, instruction-driven syntax block. By standardizing string

manipulation and mapping it to structural coordinates, Fracta serves as a predictable intermediate representation for rendering engines, plotting hardware, and algebraic compilers.

2. Lexical Structure & Tokenization

Fracta scripts are evaluated sequentially as a stream of line-buffered string arrays. Interpreters **MUST** adhere to the following parsing rules:

- **Case Sensitivity:** Structural directives (AXIOM, RULE, ANGLE, ITER) **MUST** be declared in uppercase. Variable parameters within rewrite strings are case-sensitive.
- **Line-Oriented Boundaries:** Each line in a Fracta payload represents a single, distinct compiler instruction. Arbitrary line breaks within a single directive boundary are prohibited.
- **Whitespace Treatment:** Leading and trailing horizontal whitespaces (ASCII 0x20, 0x09) **MUST** be stripped prior to execution analysis. Multiple contiguous spaces within rule transformations are compressed or treated as delimiter padding.
- **Comment Truncation:** Any line initiating with the hash character # (ASCII 0x23) or any inline string following a # sequence **MUST** be discarded by the lexical scanner as non-operational commentary space.

3. Core Directives & Structural Syntax

A valid Fracta program configures the string-rewriting engine using four foundational directives before executing a build pipeline.

3.1. AXIOM

Defines the initial seed string (the structural foundation) at recursion iteration zero ($n = 0$).

- **Syntax:** AXIOM <string>
- **Constraints:** **MUST** contain at least one valid character symbol.

3.2. RULE

Declares an production transformation rule mapping a single character predecessor to a replacement string successor.

- **Syntax:** RULE <predecessor> -> <successor>
- **Constraints:** The predecessor **MUST** be a single ASCII alphanumeric or symbolic character. The transformation delimiter -> **MUST** be present.

3.3. ANGLE

Specifies the absolute angular offset step in degrees applied during turtle rotational operations.

- **Syntax:** ANGLE <float>
- **Behavior:** When a rotation token is encountered in the compiled string, the internal heading matrix is re-computed using this step size.

3.4. ITER

Controls the total evaluation recursion depth (n).

- **Syntax:** ITER <integer>
- **Constraints:** Must be a non-negative integer. An iteration of 0 returns the un-mutated Axiom string.

4. Turtle Graphics Alphabet & Execution Semantics

Once the rewrite pipeline processes the strings to the depth specified by ITER, the resulting character chain is parsed as sequential commands for a spatial vector tracker (the "Turtle"). Compliant interpreters MUST map the following tokens to their exact physical sub-routines:

Token	Operational Command	Execution State Modification
`F`	Move Forward (Draw)	Advances the linear position coordinate from (x, y) to (x', y') drawing a structural line vector.
`G`	Move Forward (Draw)	Synonymous with `F`. Used primarily to handle dual-variable simultaneous substitutions.
`f`	Move Forward (Pen Up)	Advances the position coordinate to (x', y') without writing data to the rendering matrix.
`+`	Rotate Right	Modifies the heading angle: $\theta' = \theta + \text{ANGLE}$
`-`	Rotate Left	Modifies the heading angle: $\theta' = \theta - \text{ANGLE}$
`[`	Push State	Saves current position (x, y) and heading θ onto the LIFO tracking stack.
`]`	Pop State	Restores position (x, y) and heading θ from the apex of the LIFO tracking stack.

All non-reserved tokens (e.g., X, Y) encountered during the visual execution pass MUST be evaluated as non-operational placeholders (NOP). They alter neither the coordinate grid nor the vector state, serving strictly as structural anchors for string substitution.

5. Formal Grammar (EBNF)

The formal syntactic construct of Fracta v1.0 is governed by the following Extended Backus-Naur Form rules:

```

<fracta_script> ::= { <statement> <newline> } [ <render_cmd> ]
<statement>      ::= <comment> | <axiom_def> | <rule_def> | <angle_def> | <iter_def>

<axiom_def>      ::= "AXIOM " <string_payload>
<rule_def>       ::= "RULE " <char_token> " -> " <string_payload>
<angle_def>      ::= "ANGLE " <float_val>
<iter_def>       ::= "ITER " <int_val>
<render_cmd>     ::= "RENDER"

<string_payload> ::= { <char_token> }
<char_token>     ::= [A-Za-z0-9+\-\[\]\]
<float_val>      ::= [0-9]+ [ "." [0-9]+ ]
<int_val>        ::= [0-9]+
<comment>        ::= "#" { <any_character> }

```

6. Reference Syntax Blueprint

The following implementation example displays a valid Fracta payload configuring a **Heighway Dragon Curve** structure using a dual-state rule node system:

```

# Configuration for the Heighway Dragon Curve
AXIOM FX
RULE X -> X+YF+
RULE Y -> -FX-Y
ANGLE 90
ITER 10

# Execute processing stream
RENDER

```

7. Runtime State Machine & Memory Layout

A compliant Fracta processor MUST initialize and execute inside a deterministic 3-tier computing phase pipeline:

```

+-----+
| 1. Lexical Scanner Block |
| Parses lines -> Populates Axiom, Angle, Rules map, Iter |
+-----+
|           |
|           v           |
+-----+
| 2. Production Expansion Loop |
| Performs string rewriting bounded by iteration depth |
+-----+
|           |
|           v           |
+-----+
| 3. Turtle Trajectory Matrix |
| Evaluates characters -> Mutates LIFO Stack & Points |
+-----+

```

State Variables Memory Map

During Phase 3, the compiler context MUST track a state vector defined by the schema tuple:

$$S = \langle x, y, \theta, \mathbf{Stack} \rangle$$

Where:

- $x, y \in \mathbb{R}$: The active Cartesian coordinate matrices.
- θ : The radial heading orientation initialized at 90.0° (absolute vertical orientation).
- Stack : A Last-In, First-Out sequence handling element groupings of $\langle x_i, y_i, \theta_i \rangle$. Stack underflows caused by un-matched `]` tokens MUST throw an explicit runtime validation error.

Results

Snowflake

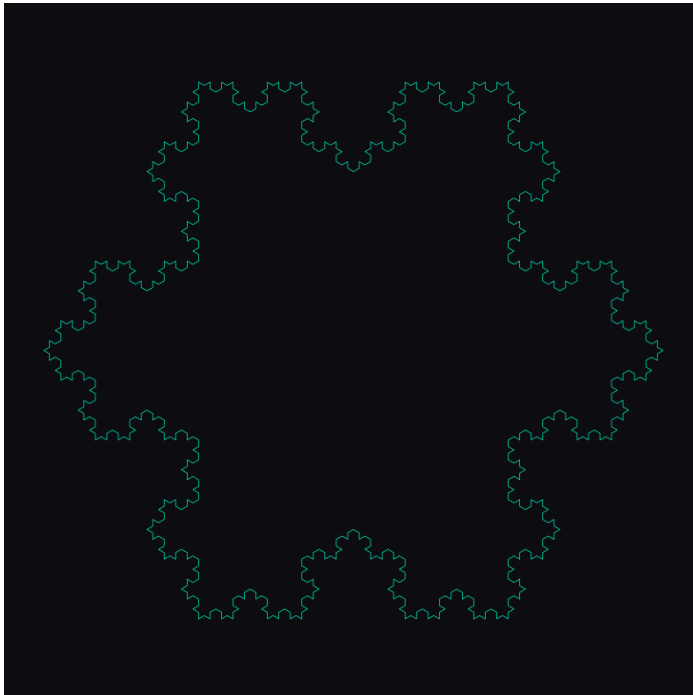


Figure: Fractal Snowflake

```
ENGINE L_SYSTEM
AXIOM F--F--F
RULE F -> F+F--F+F
ANGLE 60
ITER 4
RENDER
```

Mandelbrot Set

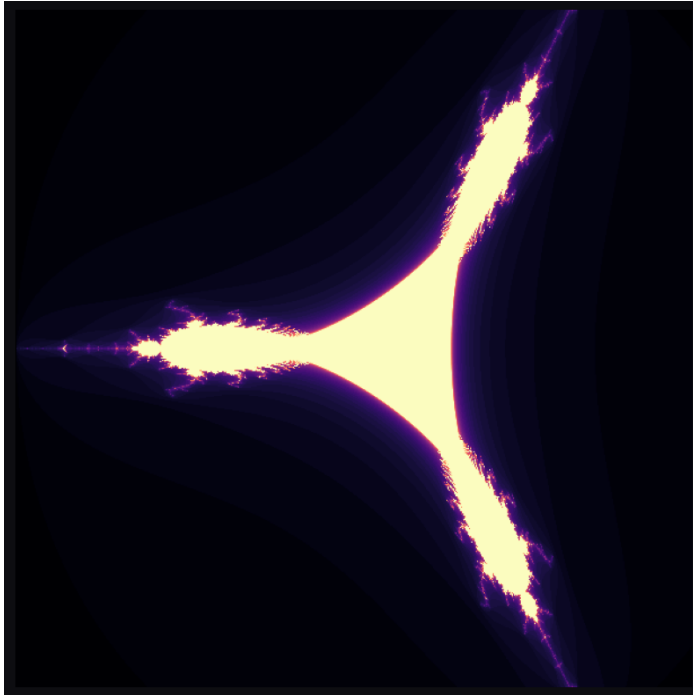


Figure: Fractal Mandelbrot Set

```
ENGINE PIXEL  
FORMULA np.conj(z)**2 + c  
X_RANGE -2.0 1.5  
Y_RANGE -1.5 1.5  
RES 600  
ITER 60  
COLORMAP magma  
RENDER
```

Fern



Figure: Fractal Fern

```
ENGINE IFS
ITER 80000
# Regole: prob, a, b, c, d, e, f
RULE 0.01  0.0  0.0  0.0  0.16  0.0  0.0
RULE 0.85  0.85  0.04 -0.04  0.85  0.0  1.6
RULE 0.07  0.2  -0.26  0.23  0.22  0.0  1.6
RULE 0.07  -0.15  0.28  0.26  0.24  0.0  0.44
RENDER
```